GTM-IP

Application Note AN012
ATOM Flexible PWM generation

Date: 21.02.2014
(Released 21.02.2014)

Robert Bosch GmbH
Automotive Electronics (AE)

LEGAL NOTICE

Revision History

Issue	Date	Remark
0.1	29.08.2011	Initial version
0.2	16.01.2012	Updated reference to GTM-IP Specification v1.5.1
0.3	02.2014	Redesign; Specification v1.5.5

Tracking of major changes

Changes between revision 1.x and 1.y

NA

Conventions

The following conventions are used within this document.

ARIAL BOLD CAPITALS	Names of signals
Arial bold	Names of files and directories
`Courier bold`	Command line entries
`Courier`	Extracts of files

References

This document refers to the following documents.

Ref	Author(s)	Title
1	AE/EIN2	GTM-IP Specification v1.5.5

Terms and Abbreviations

This document uses the following terms and abbreviations.

Term	Meaning
AEI	AE-Interface (generic bus)
AFD	AEI to FIFO Data Interface
AGC	ATOM Global Control
ARU	Advanced Routing Unit
ATOM	ARU-connected Timer Output Module
CMU	Clock Management Unit
F2A	FIFO to ARU Unit
GTM	Generic Timer Module
PSM	Parameter Storage Module
SOMC	ATOM Signal Output Mode Compare
SOMP	ATOM Signal Output Mode PWM
SYS_CLK	System Clock
TBU	Time Base Unit

Table of Contents

1 Overview

This application note describes the flexible generation of PWM signals which means that the PWM characteristics are modified on the fly by the CPU.
The GTM allows implementing this requirement with minimal CPU interaction. The CPU only has to refresh the characteristics when required. This is achieved by using the GTM internal infrastructural system that can control the GTM-IP outputs in a semi autonomous mode. If the PWM characteristics stay the same during the whole application time, the GTM-IP can work fully standalone after an initial setup by the CPU.

1.1 Use Case

In this application note the flexible PWM generation is demonstrated with two related PWM signals. These are set up in a way that their wave form is identical in period and duty cycle, only their phase is shifted. The signals can be described as follows: At first, both signals are configured with a constant duty cycle. Both signals are synchronized. To switch from the start-up phase to the phase with flexible PWM generation a third signal is used to control the other signals. The modified characteristics are provided by the CPU in a regular scheme, triggered by the mentioned control signal.

The initial duty cycle for both signals is set to 50% whereas the changing duty cycle length ranges from 0% to 100% with a resolution of 100ns. For this use case, the period length is chosen to be 10us. The control signal appears 8us after start of the last PWM period and lasts for 100ns. The duty cycle level should be defined as high for the PWM signals. The control signal level is high (low for the alternative version).
In the 10th PWM period, another signal is generated on a control channel with a certain delay to the start of the PWM period.

1.2 System architecture

For the flexible PWM generation several GTM-IP submodules are needed which are set up by the CPU and can run independently further on. The overall system is shown in **Figure 1.1**.

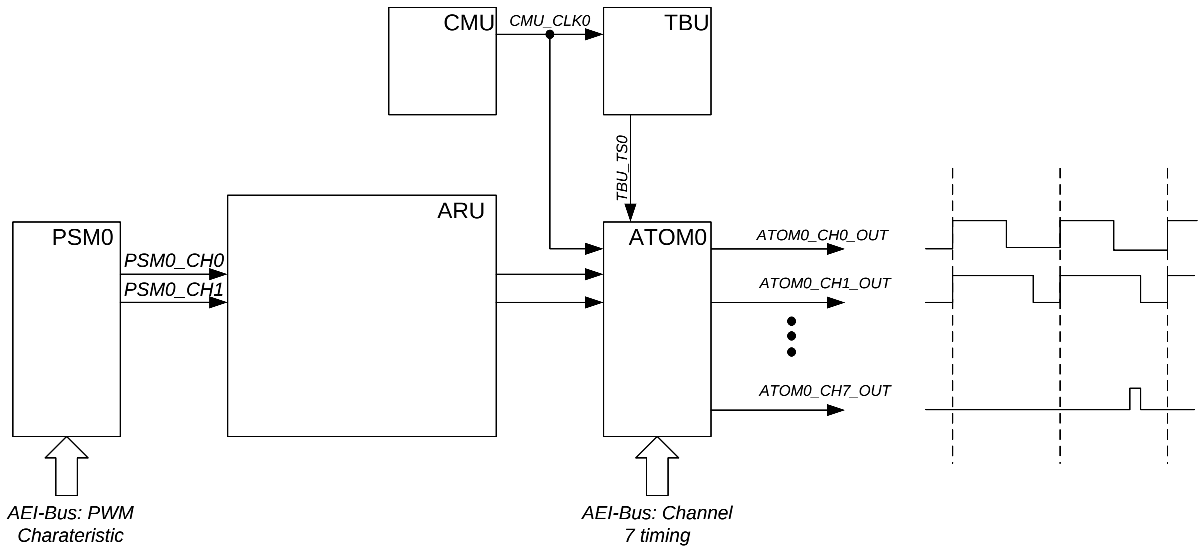

Figure 1.1: Flexible PWM generation with the ATOM submodule.

The CMU generates the clock prescalers for the PWM counter and the TBU. The PWM duty cycle parameters are loaded into the PSM channels 0 and 1 by the CPU. The F2A-Bridge within the PSM is configured to provide the FIFO data only at the high word of the ARU word. The ATOM channels 0 and 1 receive the PWM duty cycles from the ARU and generate a PWM output signal. ATOM channel 7 is used to generate one output signal in the tenth PWM period. This is done on behalf of *TBU_TS0* time base values. ATOM channel 6 is used to demonstrate an alternative implementation of the control signal (C-code only).

2 GTM Module setup

This chapter describes the necessary GTM-IP submodule configurations to establish the aforementioned application example.

2.1 CMU Setup

The CMU provides clock prescalers for the GTM-IP internal counters. For this application the CMU provides the clock for the PWM counters inside of the ATOM channels and for the system time base.
The PWM generation should have a resolution of 100ns. It is assumed, that the system clock SYS_CLK runs with 100MHz. For sake of simplicity the time base TBU is also set up with a resolution of 100ns. Therefore, only the global clock divider and CMU_CLK0 have to be configured. The configuration is shown in the following code sequence 2.1 from `init_cmu_an012()`:

```
1   CMU_GCLK_NUM = 0xFFFFFF;
2   CMU_GCLK_DEN = 0xFFFFFF;
3
4   CMU_CLK_0_CTRL = 0x9;
5
6   CMU_CLK_EN = 0x2;
```

Code 2.1: CMU Setup.

The global clock divider is configured to not divide the SYS_CLK, since a CMU_CLK0 of a 100ns resolution is needed, see line 1 and 2. Therefore, the CMU_CLK0 clock prescaler is configured to divide the SYS_CLK by a factor of ten in line 4. As a last step CMU_CLK0 is enabled in line 6.

Please note that to enable the CMU channels the GTM-IP double bit enable/disable mechanism has to be applied on the CMU_CLK_EN register. For a detailed description please refer to the corresponding submodule specification.

2.2 TBU Setup

The TBU provides common time bases for the GTM-IP. For this application the TBU is used to generate the PWM starting time for synchronization of the different PWM channels and to generate an associated signal in a sequence of PWM signals.

As shown in line 1 in code sequence 2.2 from `init_tbu_an012()` TBU channel 0 is used for this task. This channel is configured to run with normal resolution, which means that the lower 24 bits are provided to the system. As clock prescaler the CMU_CLK0 input is used. As a last step TBU channel 0 is enabled in line 3.

```
1   TBU_CH0_CTRL = 0x0;
2
3   TBU_CHEN = 0x2;
```

Code 2.2: TBU Setup.

Please note that to enable the TBU channels the GTM-IP double bit enable/disable mechanism has to be applied on the TBU_CHEN register. For a detailed description please refer to the corresponding submodule specification.

2.3 PSM Setup

The PSM stores the PWM duty cycle durations of a bunch of PWM periods before the CPU reloads new duty cycle values. For the two PWM signals, two PSM channels have to be set up.

A PSM consists of three subunits, the AFD, the FIFO and the F2A.
The AFD subunit is the interface to the CPU to provide the data, so nothing has to be configured there.
The FIFO has several configuration registers, where the depth of the FIFO, the operation mode and several interrupt possibilities can be configured and enabled. Especially, the operation mode can be of importance for a PWM generation, when the PWM pattern has to be repeated during runtime. Then the FIFO Ring Buffer Mode can be configured, where the FIFO will provide the same amount of data again and again. This application is intended to demonstrate how different pattern at the ATOM outputs are possible. Thus the Ring Buffer Mode is left disabled. Also, for sake of simplicity the FIFO depth can be kept as is. Therefore, nothing has to be configured inside of the FIFO subunit.
The F2A unit is the bridge between the FIFO and the ARU and has to provide new data to the ARU in time if requested.

The PSM submodule configuration is shown in code sequence 2.3 from `init_psm_an012()`. As stated before only the F2A unit has to be configured. In line 1 and 2 channel 0 and channel 1 are set up to provide 24 bit of data on the upper ARU word. As a last step both channels are enabled in line 4

```
1   F2A0_CH0_STR_CFG = 0x50000;
2   F2A0_CH1_STR_CFG = 0x50000;
3
4   F2A0_ENABLE = 0xA;
```

Code 2.3: PSM Setup.

Since an ARU word can have 53 bits, the F2A can be configured how the 29 bit wide FIFO data is mapped onto the ARU word. Since the PSM only stores the duty cycle durations and the ATOM expects the duty cycle duration in the upper ARU word, the F2A has to be configured to fetch data from the FIFO unit and store it in the upper internal register location for transfer to the ARU.

Please note that to enable the F2A channels the GTM-IP double bit enable/disable mechanism has to be applied on the F2A_ENABLE register. For a detailed description please refer to the corresponding submodule specification.

2.4 ATOM setup

The PWM signals and one associated control signal should be generated with the ATOM submodule on the channels 0 and 1 (PWM signals) and the ATOM channel 7 associated signal (ATOM channel 6 respectively). The idea is to run the two PWM signals with the ATOM SOMP mode and to control the associated signal output timing with the ATOM SOMC mode. The PWM period is provided to the ATOM via the CPU at system startup, while the duty cycles are received via ARU. The associated signal timing is provided by the CPU after the FIFOs are filled with the duty cycle parameters

There are several steps necessary to set up the ATOM channels. This configuration has to be done in advance. It has to be decided if the PWM generation on the two channels should be done with separate counters or if the first ATOM channel counter should be used. For the present application independent counters are chosen. The duty cycle level is set to high. In addition, the ATOM channel 0 and 1 counters can be preloaded to define a specific offset for the PWM signals. Here, the signals should be shifted by 50% of the period which means that the second signal starts later than the first signal (line 4 and 12).

As mentioned above, the control channel(s) should generate an associated signal at a specific point in time, which consists of a rising edge and a falling edge 100ns later. Therefore, the SOMC mode is needed with a SL bit set to '1' to start with a low ATOM channel output.

For a continuous update of the PWM duty cycle values an event has to be generated which informs the CPU to reload the FIFOs and also the control channel's compare registers. This is implemented by an interrupt coming from the control channel's capture compare unit.

After the channels were configured, the AGC is set up to control the enabling of the ATOM channels. As mentioned above, the channels should be enabled on behalf of a specific time stamp value. A forced update is necessary before enabling the update mechanism (line 29f). Otherwise channel 1 would run with system clock instead of the CMU generated clock for the first cycles and the phase shift would not be correct.

The overall code sequence for the mentioned configuration steps is shown in code sequence 2.4 from `init_atom_an012()`. It is important to enable the outputs as a last step. Otherwise the output would show intermediate states of the internal ATOM structures and the signals would not have the expected level at starting time.

```
0    #define PWM_PERIOD 100       //define PWM period to 10us (100*100ns)
1
2  // setup ATOM channels 0 in SOMP
3  ATOM0_CH0_RDADDR = F2A0_WRADDR0;      // use data from PSM0 channel 0
4  ATOM0_CH0_CN0 = 99;                   // start immediately with PWM
5  ATOM0_CH0_CM0 = PWM_PERIOD;           // set PWM period
6  ATOM0_CH0_SR0 = PWM_PERIOD;           // set PWM period
7  ATOM0_CH0_CTRL = 0x82A;               // SL=1, CMU_CLK0, ARU_EN=1,
8                                        // load ARU high word, SOMP
9
```

```
// setup ATOM channel 1 in SOMP
ATOM0_CH1_RDADDR = F2A0_WRADDR1;      // use data from PSM0 channel 1
ATOM0_CH1_CN0 = 49;                   // start 50% later than ch0 PWM
ATOM0_CH1_CM0 = PWM_PERIOD;           // set PWM period
ATOM0_CH1_SR0 = PWM_PERIOD;           // set PWM period
ATOM0_CH1_CTRL = 0x82A;               // SL=1, CMU_CLK0, ARU_EN=1,
                                      // load ARU high word, SOMP

// setup ATOM channel 7 in SOMC and enable CCU1TC interrupt
ATOM0_CH7_CTRL = 0x921;               // SL=1, ACB42=100, ARU_EN=0, SOMC
ATOM0_CH7_IRQ_EN = 0x2;               // enable CCU1TC IRQ
ATOM0_CH7_CM0 = 1280;                 // rising edge 108us after 1st PWM
ATOM0_CH7_CM1 = 1281;                 // falling edge 100ns later

// alternatie: setup ATOM channel 6 in SOMC with different settings
ATOM0_CH6_CTRL = 0x121;               // SL=0, ACB42=100, ARU_EN=0, SOMC
ATOM0_CH6_CM0 = 1280;                 // rising edge 108us after 1st PWM
ATOM0_CH6_CM1 = 1281;                 // falling edge 100ns later

// setup AGC
ATOM0_AGC_FUPD_CTRL = 0x0000A;       // force update on ch0 and 1
ATOM0_AGC_GLB_CTRL  = 0xA0001;        // enable update mechanism for
                                      // ch. 1, 0 & host trigger to
                                      // trigger forced update

ATOM0_AGC_ENDIS_CTRL = 0xA00A;        // enable ch. 7, 6, 1, 0

ATOM0_AGC_ACT_TB = 0x010000C8;        // set up timebase cmp 200

ATOM0_AGC_OUTEN_CTRL = 0xA00A;        // enable ATOM output
                                      // channel 7, 6, 1, 0
```

Code 2.4: ATOM Setup.

Please note that to enable the ATOM channels and outputs the GTM-IP double bit enable/disable mechanism has to be applied on the corresponding registers. For a detailed description please refer to the corresponding submodule specification.

2.5 Application setup

This section describes the interaction of the main thread and the ISR which implements the flexible PWM generation. The relevant code is shown in code sequence 2.5.

The submodule configuration is done step by step in the main thread `sw_atom_flex_pwm_an012()` using the subfunctions described earlier (line 1-4). Furthermore the first bunch of PWM duty cycle durations is provided to the PSM channels 0 and 1 (line 6-10).

Any subsequent refill of the FIFOs, the reset of the interrupt notification and the setup of the next capture/compare values for the ATOM channel 7 is done within an ISR called `sw_atom_flex_pwm_an012_isr()` (line 21-41).

To simplify the task of providing new PWM characteristics the same bunch of values is provided several times in a loop. Nevertheless the values are actually written to the FIFO (line 25-28), so the mechanism is as intended.

```
 1    init_cmu_an012();
 2    init_tbu_an012();
 3    init_psm_an012();
 4    init_atom_an012();
 5
 6    // provide initial PWM duty cycles (50%): fill PSM channels
 7    for (int i=0; i<10; i++) {
 8          AFD0_CH0_BUF_ACC = 50;
 9          AFD0_CH1_BUF_ACC = 50;
10    }
//////////////////////////////////////////////////////////////////////
21    int mp = 0; // matching time of compare
22    int dummy = 0;
23
24    // refill PSM with new PWM duty cycles
25    for (int i=0; i<11; i++) {
26        AFD0_CH0_BUF_ACC = i*10;  // generate PWM from 0% to 100%
27        AFD0_CH1_BUF_ACC = i*10;  // generate PWM from 0% to 100%
28    }
29
30    // clear IRQ and set up new capture/compare values for channel 7
31    ATOM0_CH7_IRQ_NOTIFY = 0x2;   // clear IRQ notify bit
32
33    mp = ATOM0_CH7_SR0;           // read match time,
34                                  // enables cap/com again!
35    dummy = ATOM0_CH6_SR0;        // dummy read match time,
36                                  // enables cap/com again!
37
38    ATOM0_CH7_CM0 = mp + 1079;    // set up new match event
39    ATOM0_CH7_CM1 = mp + 1080;    // set up new match event
40    ATOM0_CH6_CM0 = mp + 1079;    // set up new match event
41    ATOM0_CH6_CM1 = mp + 1080;    // set up new match event
```

Code 2.5: Flexible PWM application.

3 APPENDIX

3.1 C-Code

```
//---------------------------------------------------------------------------
// Project Name : GTM - Generic Timer Module
//---------------------------------------------------------------------------
// Copyright    : Robert Bosch GmbH, Gerlingen, Germany
// Department   : AE/EIY
// Created      : 30.01.2014
//---------------------------------------------------------------------------
//:head----------------------------------------------------------------------
//---------------------------------------------------------------------------
// Description       :   sw_atom_flex_pwm_an012
//                       implements
//                       Application Note AN012
//                       ATOM Flexible PWM generation
//===========================================================================
// Important Notes :
//===========================================================================

#include "gtm.h"
#include "gtm_device_cfg.h"
#include "gtm_defs.h"
#include "functions.h"
#include <cstdlib>

//---------------------------------------------------------------------------
// init functions

void init_cmu_an012(void)
{

    // setup global clock divider
    CMU_GCLK_NUM = 0xFFFFFF;
    CMU_GCLK_DEN = 0xFFFFFF;

    // setup the CMU_CLKx prescalers
    CMU_CLK_0_CTRL = 0x9; // divide SYS_CLK by 10

      // enable clock prescalers (CMU_CLK0 only)
    CMU_CLK_EN = 0x2;

}  //init_cmu_an012()
```

```
void init_tbu_an012(void)
{

    // setup TBU channel 0
    TBU_CH0_CTRL = 0x0; // no LOW_RES, chose CMU_CLK0

   // enable TBU channel 0
   TBU_CHEN = 0x2;

}  //init_tbu_an012()


void init_psm_an012(void)
{

    // setup FIFO not needed for this application
    // setup F2A channels 0 and 1 to provide 24 bit of data on the upper
    // ARU word
    F2A0_CH0_STR_CFG = 0x50000; // transfer high word, FIFO ->□ ARU
    F2A0_CH1_STR_CFG = 0x50000; // transfer high word, FIFO □-> ARU

    // enable F2A channels 0 and 1
    F2A0_ENABLE = 0xA;

}  //init_psm_an012()

void init_atom_an012(void)
{

      #define PWM_PERIOD 100       //define PWM period to 10us (100*100ns)

    // setup ATOM channels 0 in SOMP
    ATOM0_CH0_RDADDR = F2A0_WRADDR0;     // use data from PSM0 channel 0
    ATOM0_CH0_CN0 = 99;                  // start immediately with PWM
    ATOM0_CH0_CM0 = PWM_PERIOD;          // set PWM period
    ATOM0_CH0_SR0 = PWM_PERIOD;          // set PWM period
    ATOM0_CH0_CTRL = 0x82A;              // SL=1, CMU_CLK0, ARU_EN=1,
                                         // load ARU high word, SOMP

    // setup ATOM channel 1 in SOMP
    ATOM0_CH1_RDADDR = F2A0_WRADDR1;     // use data from PSM0 channel 1
    ATOM0_CH1_CN0 = 49;                  // start 50% later than ch0 PWM
    ATOM0_CH1_CM0 = PWM_PERIOD;          // set PWM period
    ATOM0_CH1_SR0 = PWM_PERIOD;          // set PWM period
    ATOM0_CH1_CTRL = 0x82A;              // SL=1, CMU_CLK0, ARU_EN=1,
                                         // load ARU high word, SOMP

    // setup ATOM channel 7 in SOMC and enable CCU1TC interrupt
    ATOM0_CH7_CTRL = 0x921;              // SL=1, ACB42=100, ARU_EN=0, SOMC
    ATOM0_CH7_IRQ_EN = 0x2;              // enable CCU1TC IRQ
      ATOM0_CH7_CM0 = 1280;              // rising edge 108us after 1st PWM
    ATOM0_CH7_CM1 = 1281;                // falling edge 100ns later
    // alternatie: setup ATOM channel 6 in SOMC with different settings
    ATOM0_CH6_CTRL = 0x121;              // SL=0, ACB42=100, ARU_EN=0, SOMC
    ATOM0_CH6_CM0 = 1280;                // rising edge 108us after 1st PWM
    ATOM0_CH6_CM1 = 1281;                // falling edge 100ns later

    // setup AGC
    ATOM0_AGC_FUPD_CTRL = 0x0000A;       // force update on ch0 and 1
    ATOM0_AGC_GLB_CTRL  = 0xA0001;       // enable update mechanism
                                         // for ch. 1, 0 & host trigger
                                         // to trigger forced update

    ATOM0_AGC_ENDIS_CTRL = 0xA00A;       // enable ch. 7, 6, 1, 0

    ATOM0_AGC_ACT_TB = 0x010000C8;       // set up timebase cmp to 200

    ATOM0_AGC_OUTEN_CTRL = 0xA00A;       // enable ATOM output
                                         //channel 7, 6, 1, 0

}  //init_atom_an012()


//---------------------------------------------------------------------------
// main thread:
int sw_atom_flex_pwm_an012(void)
{
    if ((PSM_INSTANCES  > 0) &&
            (ATOM_INSTANCES > 0) )
      {
            // initialize and enable all modules
            init_cmu_an012();
            init_tbu_an012();
            init_psm_an012();
            init_atom_an012();

            // provide initial PWM duty cycles (50%): fill PSM channels
            for (int i=0; i<10; i++) {
                  AFD0_CH0_BUF_ACC = 50;
                  AFD0_CH1_BUF_ACC = 50;
            } // for
      } // if

    return 0;

} // sw_atom_flex_pwm_an012()
```

```
//---------------------------------------------------------------------------
// ISR

void sw_atom_flex_pwm_an012_isr(int isr_nr)
{

    int mp = 0; // matching time of compare
      int dummy = 0;

      // refill PSM with new PWM duty cycles
    for (int i=0; i<11; i++) {
        AFD0_CH0_BUF_ACC = i*10;    // generate PWM from 0% to 100%
        AFD0_CH1_BUF_ACC = i*10;    // generate PWM from 0% to 100%
    }

      // clear IRQ and set up new capture/compare values for channel 7
    ATOM0_CH7_IRQ_NOTIFY = 0x2;     // clear IRQ notify bit

    mp = ATOM0_CH7_SR0;             // read match time,
                                    // enables cap/com again!
    dummy = ATOM0_CH7_SR0;          // dummy read match time,
                                    // enables cap/com again!

    ATOM0_CH7_CM0 = mp + 1079;      // set up new match event
    ATOM0_CH7_CM1 = mp + 1080;      // set up new match event
    ATOM0_CH6_CM0 = mp + 1079;      // set up new match event
    ATOM0_CH6_CM1 = mp + 1080;      // set up new match event

} //atom_flex_pwm_an12_isr()
```